EDMS 755349
July 11, 2006
Version 3.0

TELL1 VHDL Framework User Manual

Guido Haefeli[1]

Abstract

TELL1 is an FPGA based readout board for LHCb. A large fraction of the design as synchronization to the TTC and the read out network interface are identical for all sub-detector allowing to re-use a large fraction of the design. For the data acquisition of the various sub-detectors, the FPGA firmware implementation for the processor FPGAs needs to be adapted to the specific requirements. A VHDL framework has been developed to support a common platform for the specific implementations. The present document is a user guide for installing and updating to the latest framework version and shows how to add user specific designs.

To be approved by:	
To be checked by:	Guido Haefeli
	Hou Lei
	Alex Gong
	Manfred Muecke

[1]E-mail: Guido.Haefeli@epfl.ch Home Page http://lphe.epfl.ch/ ghaefeli

Contents

1 Revision

Version 3.0, 21.6.2006
New version without SDRAM and new framework layout (it has changed quite a bit since 1.5 years!)
Version 2.4, 22.1.2005

2 Intorduction

A large amount of VHDL programming has been done to implement the FPGA functionality on the TELL1 board. This guide should help the users during installation and development for the specific needs of the different sub-detectors. Please report any problems concerning the installation to Guido.Haefeli@epfl.ch in order to improve and facilitate the task for new users.
A mailing list for the TELL1 VHDL developer has been created to supply the interested people with news and issues during the development. Please sign you on the list to get the latest information directly, here the links to the mailing list and to subscribe to mailing list Mailing list, Subscribe. The latest release of the VHDL Framework can be found on the TELL1 Home Page. There you will also find a lot of documentation and useful links the TELL1 related topics.

3 Quick guide for TELL1 Framework installation

If you think you will not need any detailed instructions because you know already well how to deal with the TELL1 framework you should just pass through the following steps to get the setup done:

- Install the Mentor Graphics FPGA Advantage (see 6.1 for version), choose all standard options for the installer (you can choose either option HDL Designer or HDL Author!)
- Copy the Tool settings for the HDL Designer into your application data directory (see section 6.3)
- Edit the batch file for setting the necessary System Environment Variables. The file is located in
  `<hdl_dir>\framework_setup`
  (see section 6.2).
- Check that the synthesis invocation runs the user Tcl script for the data and version setting at the start of synthesis. This is done in HDL Designer ⇀ Settings ⇀ Task Window ⇀ *right click* Precision Synthesis Icon ⇀ Tab User Script

`../../../../CLI_compile/version_precision.tcl`
should be called at the start of the synthesis (see Fig. 2)

- Run the Quartus project assignments for your sub-detector design, located in `<hdl_dir>\user_tell1_vhdl_libraries\quartus_tcl`.

- After the .sof files of the SyncLink and the PP-FPGAs have been generated use Quartus conversion of programming files to generate the combined .pof file. There is an example convert file stored in
`<hdl_dir>\programmer_files\velo_pof_setup_comp.cof`
Each sub-detector should have its own "cof" file in order that the correct PP-FPGA "sof" file used. For more details see 8.

4 Quick guide for updates

You have to be aware that each time a new version of the framework is released the common and your user specific VHDL design has to be merged. The strategy of version management is to keep for each version the complete VHDL-framework (the hdl directory). Your design located in the user_tell1_vhdl_libraries directory can be inserted after updating to the new version.
The following steps are needed to make an update to a new version of the framework:

1. Make sure that your FPGA Advantage version is equal to the one used for the framework design - update the version if needed, (see 6.1).

2. Copy the complete hdl directory into your desired location.

3. Update the HDL Designer Series.zip 6.3

4. Update the library mapping file TELL1_user_project.hdp 6.5 if you have added new libraries yourself.

5. Change the environment variables if needed in set_environment_variables.bat 6.2 and execute the batch file by double clicking on it.

6. Add you custom library (user libraries) to the new hdl directory.

7. Update the interface of changed common components in HDL Designer if the generation or compilation shows some inconsistent interface errors.

5 CVS repository at Cern

To keep up with new versions of the framework software and all files being used in its context, a public accessible CVS repository has been set. Because the HDL Designer adds

and removes complete directories from the design, CVS can not easily be used for the HDL Designer source files but only for the VHDL output. These text files are checked into the repository for each release.
The c-code developed to debug and monitor the TELL1 including CCPC access, GBE sniffer and the TELL1 processing algorithm models are also on CVS. The repository can be found at: http://isscvs.cern.ch/cgi-bin/viewcvs-all.cgi/?cvsroot=TELL1

6 FPGA design development Software

It is clear that during the time of the development several changes of the software version will be necessary. It is impossible to guarantee compatibility with older software versions since the evolution of the software is very fast. Regarding the large amount of design blocks, libraries and components used in the TELL1 framework that a text only based development can not be envisaged. The design of the framework has been all written by the support of the Mentor Graphics HDL designer software. The software nevertheless allows to export the automatically generated VHDL text files. For present design of the framework can be completely copied and used with the Mentor software having the advantage of the graphical support. It has to be pointed out, that for designers that do want to use an import procedure into an other graphical environment that each time a new version of framework will be released, a new import procedure has to be done.

6.1 Current software version

Framework version	FPGA Advantage	Quartus
Up to v1.4	v6.3	5.0 + SP1
v1.5	v7.2	5.0 + SP1

Remark that the use of an older version of HDL Designer is not possible since on the pre 6.0 FPGA Advantage version the database is different. The FPGA Advantage is available for universities directly from Mentor Graphics Europe and does cost about 700 CHF including the direct support from Mentor. For more information see http://www.mentor.com/company/higher_ed/index.cfm or if this link is not valid anymore contact the local Mentor support.

6.2 Setting up the system variables

To cope with different installation directories for the framework system variables are used. During the development the design was located in `e:\hdl` and if you still find some problems where this path appears, you can report on this so it can be replaced with the

according variables. I put quite some effort to avoid it the `e:\hdl` problem!. To make the settings more convenient a script to set the variables is given in:
`<hdl_dir>\environment_setup\set_environment_variables.bat`
It takes care to setup the used variables.
Copy the script to a location where it is not overwritten by a future update of the framework, so you can use your settings again. The following variables are set (remark that the current settings on my local PC are given as an example. Customize the values in the `set_environment_variables.bat` file as appropriate for your setup!!

`Variable name`	`/ or \`	Description
`precision_exe`	`\`	Precision executable
`hdl_root_windows`	`\`	Root directory of the framework
`user_tell1_vhdl_libraries`	`\`	Contains the detector designs for PP and SyncLink-FPGA
`common_tell1_vhdl_libraries`	`\`	Common vhdl libraries used by all subdetectors
`vhdl_libraries`	`\`	Vendor specific libraries (stratix, altera_mf, lpm, sgate)
`detector_spec_vhdl_libraries`	`\`	Detector specific libraries
`vhdl_sim_data`	`\`	This is where the testbench is accessing files, this is almost replaced by the memory initialization files located in the local simulation directory work
`memory_ini`	`\`	The memory initialization files for memories, and PLL reconfigure.
`hdl_root`	`/`	The same path as at 2. for Quartus.
`user_libs_root`	`/`	User libraries

6.3 Installation of the FPGA Advantage

All necessary instructions for installation and licensing are given by Mentor Graphics and Altera. To import the tool option settings for the Mentor software the application specific data for the "HDL Designer Series" needs to be imported. The zip file containing the standard tool settings is placed in:

`<hdl_dir>\hdl_ini_files\HDL Designer Series.zip`

Copy this to the "Application Data" directory on your PC which is typically for W2K and XP at

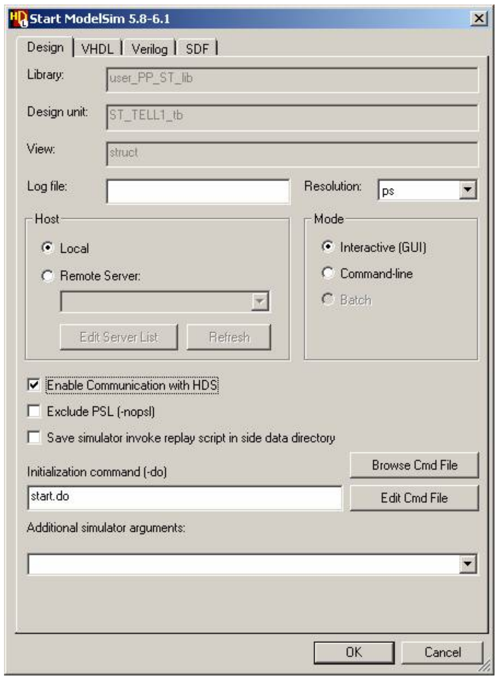
`C:\Documents and Settings\<your user name>\Application Data`

(if you do not find this directory with the Windows Explorer it is because Windows hides the system files (change the option). The directory has to be called identical as it is given above. Note that the following settings for the tool options are needed:

6.4 ModelSim and Precision invocation

By copying the HDL Designer Series application data directory (see 6.3) to your local machine, the following list of invocation settings are automatically done. You should maybe verify that this is true!

- The ModelSim simulation time resolution needs to be 1 ps **this is required for the PLL simulation models**.
- The language option for VHDL has is set to VHDL-93 since the files used for the simulation stimulus are called with this formalism and the VHDL-87 does not support the assignments for example: `data <= X"ABCD"` (usage of hexadecimal notation for vectors.
- [optional] On starting up the ModelSim simulation a script is called with the command **start.do**, where the initialization of the simulation is done. This script is stored in simulation downstream directory of each library (the work directory). So for example for the `user_PP_VELO_lib` it is in `user_PP_VELO_lib\work`. This might be helpful to add in your own test benches.

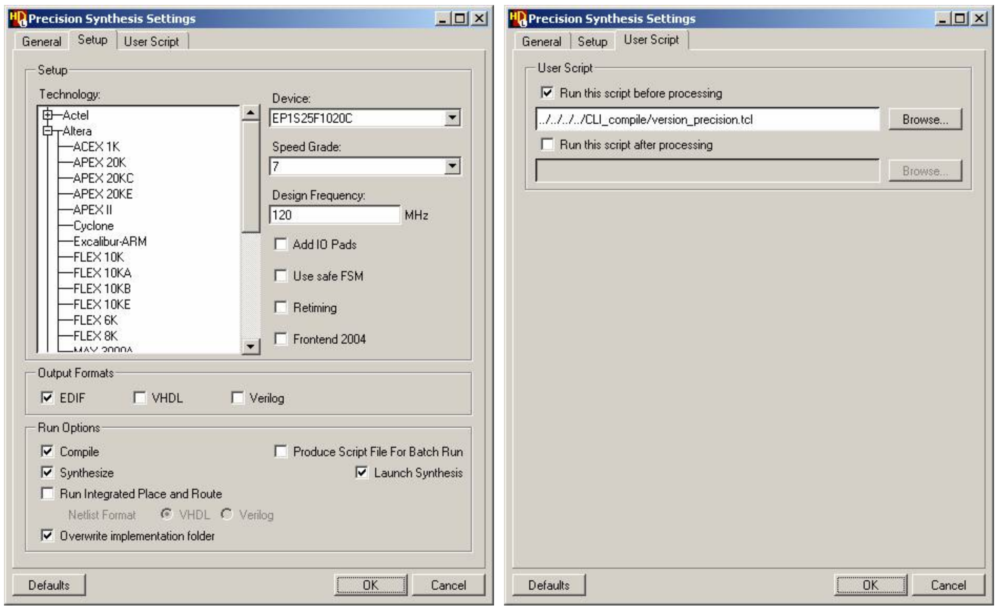


Library: user_PP_ST_lib
Design unit: ST_TELL1_tb
View: struct
Log file:
Resolution: ps
Host: [✓ Local] Remote Server:
Mode: [✓ Interactive (GUI)] Command-line Batch
[✓ Enable Communication with HDS]
Exclude PSL (-nopsl)
Save simulator invoke replay script in side data directory
Initialization command (-do): start.do
Additional simulator arguments:

Figure 1: The invocation window for Modelsim.

- [Precision is used] At synthesis invocation set the device to the Altera - Stratix - EP1S25C780 or EP1S25C1020 - 7 Speed Grade - 40 MHz.

- [Precision is used] At synthesis invocation the `ADD_IO_Pads` option is not ”which means un-set“.



Precision Synthesis Settings

General | Setup | User Script

Setup

Technology: Actel, Altera (ACEX 1K, APEX 20K, APEX 20KC, APEX 20KE, APEX II, Cyclone, Excalibur-ARM, FLEX 10K, FLEX 10KA, FLEX 10KB, FLEX 10KE, FLEX 6K, FLEX 8K, MAX 3000A)

Device: EP1S25F1020C

Speed Grade: 7

Design Frequency: 120 MHz

☐ Add IO Pads

☐ Use safe FSM

☐ Retiming

☐ Frontend 2004

Output Formats

☑ EDIF ☐ VHDL ☐ Verilog

Run Options

☑ Compile

☑ Synthesize

☐ Run Integrated Place and Route

Netlist Format ● VHDL ○ Verilog

☑ Overwrite implementation folder

☐ Produce Script File For Batch Run

☑ Launch Synthesis

Defaults | OK | Cancel

Precision Synthesis Settings

General | Setup | User Script

User Script

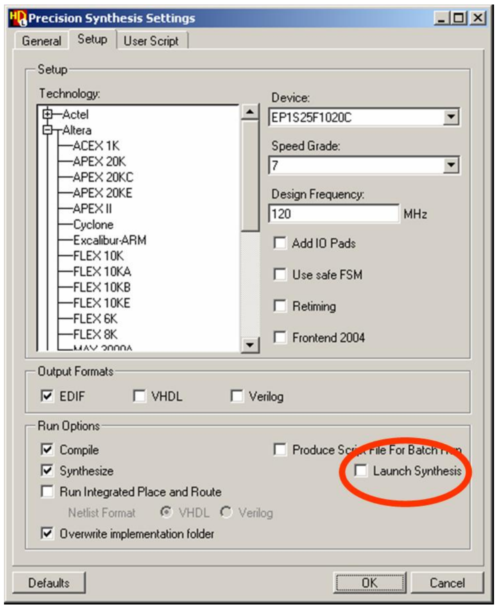
☑ Run this script before processing

../../../../CLI_compile/version_precision.tcl

☐ Run this script after processing

Browse... | Defaults | OK | Cancel

Figure 2: The invocation window for Precision.



Precision Synthesis Settings

General | Setup | User Script

Setup

Technology: Actel, Altera (ACEX 1K, APEX 20K, APEX 20KC, APEX 20KE, APEX II, Cyclone, Excalibur-ARM, FLEX 10K, FLEX 10KA, FLEX 10KB, FLEX 10KE, FLEX 6K, FLEX 8K, MAX 3000A)

Device: EP1S25F1020C

Speed Grade: 7

Design Frequency: 120 MHz

☐ Add IO Pads

☐ Use safe FSM

☐ Retiming

☐ Frontend 2004

Output Formats

☑ EDIF ☐ VHDL ☐ Verilog

Run Options

☑ Compile

☑ Synthesize

☐ Run Integrated Place and Route

Netlist Format ● VHDL ○ Verilog

☑ Overwrite implementation folder

☐ Produce Script File For Batch Run

☐ Launch Synthesis

Defaults | OK | Cancel

Figure 3: Remove the L̈aunch Synthesisöption to generate the add_files.tcl script only. This is useful if you synthesize with Quartus.

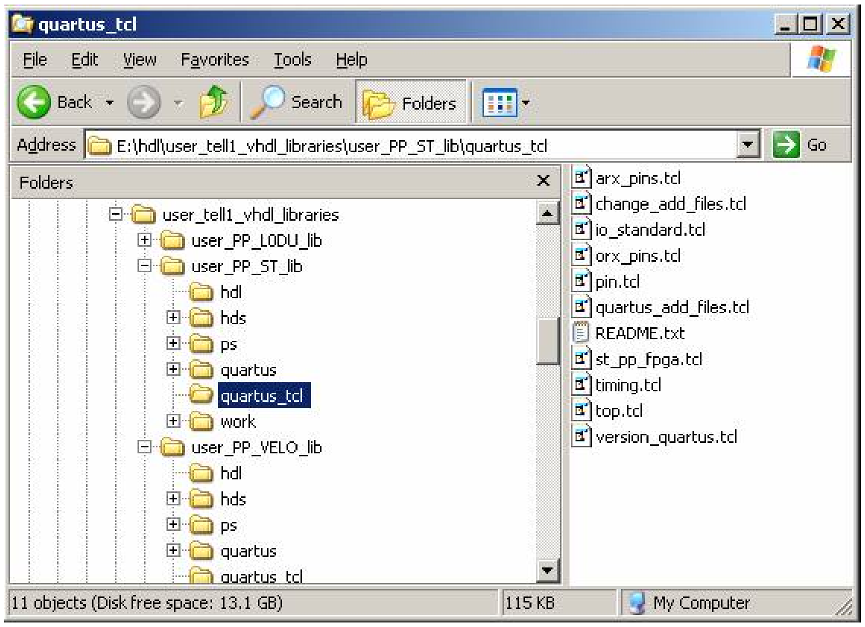


Figure 4: Tcl scripts for the PP-FPGA.

6.5 Library mapping file

To import the library mappings from the framework, the TELL1 framework project library mappings needs to be specified in the HDL Designer. The file to point to is:
`<hdl_root>\hdl_ini_files\TELL1_user_project.hdp`
You can add your own libraries to this file.

7 Quartus setup

The TELL1 VHDL framework provides a complete set of script files to setup the Quartus environment for Place and Route and for direct VHDL synthesis with Quartus.

7.1 Tcl script files for Quartus

A set of Tcl scripts is provided in order to obtain all constraints for Quartus necessary for Synthesis and Place and Route.
The scripts can be easily adapted for other sub-detectors. They are tested to be used with the environment variable settings given in 6.2. A set of scripts is copied for each sub-detector PP-FPGA and is located in:
`<hdl_root>\user_tell1_vhdl_libraries\user_PP_ST_lib\quartus_tcl` for example for the ST sub-detector (Fig. 4. The task the different scripts are given in the README.txt. There is the top.tcl script that calls in sequence the other script files that produce each

some project assignments as IO standard definitions or PIN out of the chips. In the top.tcl 6 parameters can be changed to customize the scripts for new sub-detectors. These are:

projname	Name of the top level entity eg **st_pp_fpga**
projname_struct	Name of the top level entity and architecture combined with underscore, used by the precision tool eg **st_pp_fpga_struct**.
path	Path to the hdl installation directory eg **e:\hdl**
path_user_lib	Path to the user library directory eg **e:\hdl\user_tell1_vhdl_libraries\user_PP_VELO_lib**. The directory foreseen to be used for the user libraries is the user_tell1_vhdl_libraries directory as already done for Velo, ST, L0DU, ...
A-Rx or O-Rx	Choose one of the two pin definitions of the receiver part (ST,OT,L0MUON,MUON,... choose O-Rx)
Synthesis tool	Choose to use Precision or Quartus as the synthesis too. Recommended is the Quartus!

7.1.1 Create new Quartus project

You can create a new Quartus project by running the top.tcl script. If there is no project already created in the quartus folder it will create a new one with all project assignments necessary.

Customize the parameters indicated in the top.tcl !!
The script top.tcl can be lounged on the tcl command line in Quartus. You can write: **source gen.tcl** (gen.tcl is a small script located in the quartus directory that calls the top.tcl script).

7.2 EDIF or VHDL file import

In the top.tcl you can choose by commenting out the correct lines between Quartus synthesis or EDIF file import. If the VHDL file import is chosen, a list of VHDL design files is made available and assigned to Quartus. For the EDIF import it is one single file that is imported.

8 Create TELL1 pof file

The programming file that goes in the single EEPROM located on the TELL1 needs the combined PP_FPGA and SyncLink_FPGA programmer file. Use the generated programming files for each chip (located in the Quartus project directory) to combine the two files into one. Conversion setup files for each sub-detector are located in the folder: `<hdl_root>\programmer_files`. The pof file generation is also available in the command line interface (CLI)?? where a script is calling the conversion file (.cof) and outputs the pof file into the programmer_files directory.

8.1 User and common VHDL libraries

The so called user libraries contain the design specific to the sub-detectors (users). This includes the top level design for each of the two FPGAs pp_fpga and synclink_fpga which each is accommodated in their specific libraries. In addition user specific libraries for certain tasks on the chip are foreseen. Browse the structure of the directories to get an overview:

```
<hdl_dir>\common_tell1_vhdl_libraries
<hdl_dir>\detector_spec_vhdl_libraries
<hdl_dir>\user_tell1_vhdl_libraries
```

9 Test bench

The complete TELL1 board with CCPC interface, TTC interface, FrontEnd interface, and GBE interface is given in the top level test bench. Have a look at the ST design as an example.

9.1 Test bench initialization

The test bench allows to setup all ECS resources via ECS bus using a dedicated library that models the CCPC access (see Fig. 6). This method can typically used to setup a small number of registers. ECS accesses in the ModelSim environment are very time consuming (1 μs per ECS access, resulting in about 1 s of simulation time). As soon as a complete parameter set is required, the memory initialization files are used.
The memory initialization files are generated via c-code on the CCPC based on the configuration file. This is the preferred method also to obtain a consistent setup between the VHDL test bench and the TELL1 c-code setup. Modelsim allows to import memory initialization files for every memory or register in the design. The format used is the .mem format [2] An extract of such a file is given below:

```
// memory data file (do not edit the following line - required for mem load use)
// instance=/../../..
// format=mti addressradix=h dataradix=h version=1.0
1f: 9F9F9F9F 9F9F9F9F 9F9F9F9F 9F9F9F9F 9F9F9F9F 9F9F9F9F 9F9F9F9F 9F9F9F9F
17: 9F9F9F9F 9F9F9F9F 9F9F9F9F 9F9F9F9F 9F9F9F9F 9F9F9F9F 9F9F9F9F 9F9F9F9F
 f: 9F9F9F9F 9F9F9F9F 9F9F9F9F 9F9F9F9F 9F9F9F9F 9F9F9F9F 9F9F9F9F 9F9F9F9F
 7: 9F9F9F9F 9F9F9F9F 9F9F9F9F 9F9F9F9F 10B010B0 10101010 80B080B0 10B010B0
```

The procedure required to initialize the test bench is the following:

[2] Don't confuse this with the option to initialize the memory with a .hex or .mif file via vhdl.

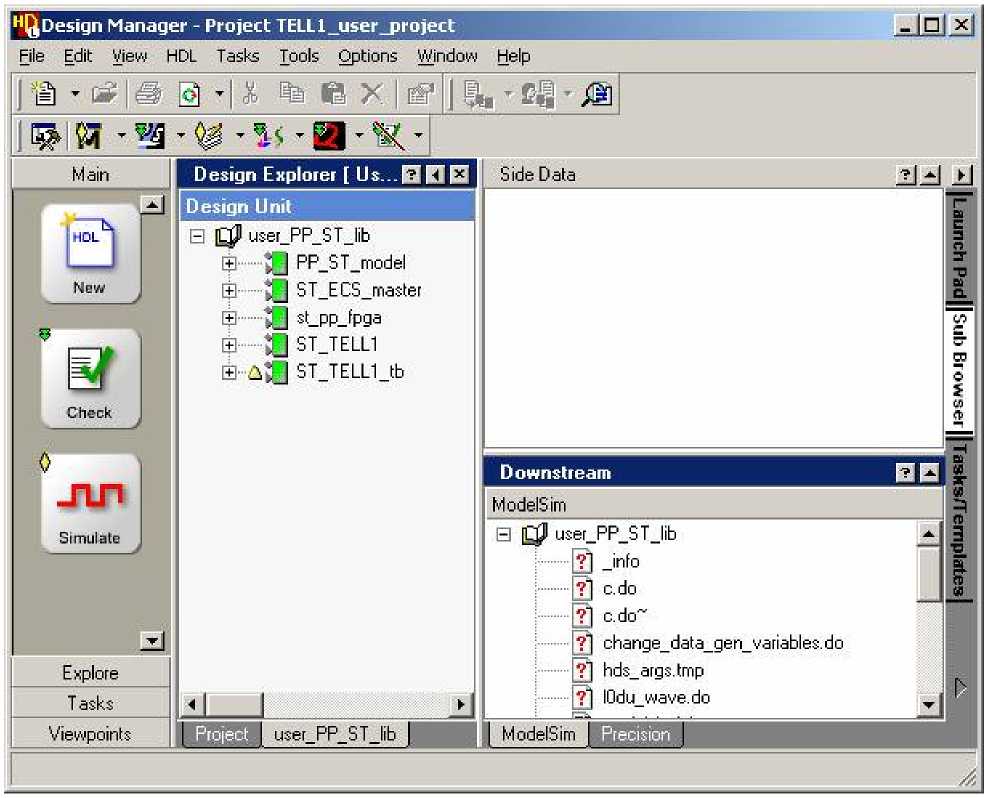
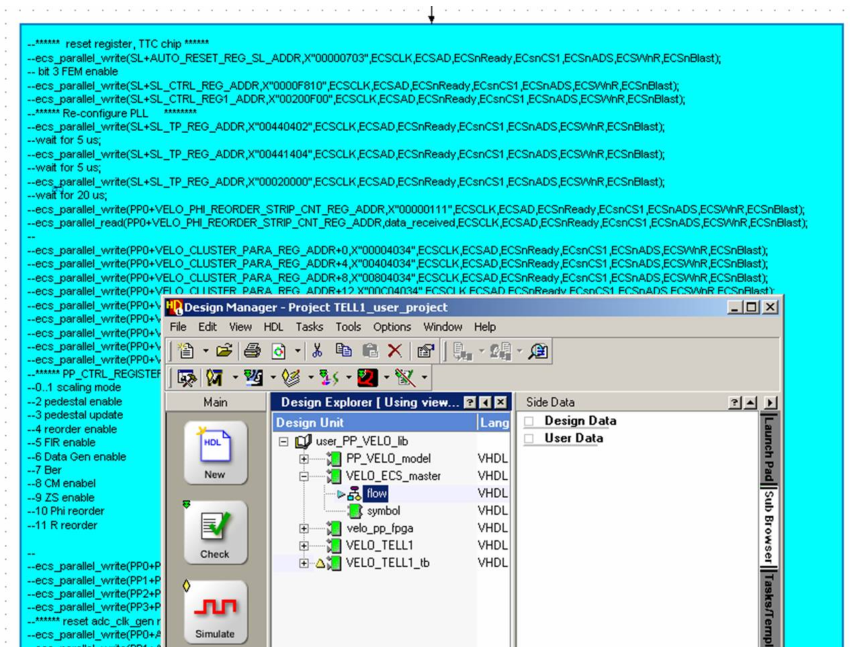


Figure 5: ST design in the user_PP_ST_lib. The ST_TELL1_tb is the top level test bench component.



```
--******  reset register, TTC chip ******
--ecs_parallel_write(SL+AUTO_RESET_REG_SL_ADDR,X"00000703",ECSCLK,ECSAD,ECSnReady,ECsnCS1,ECSnADS,ECSWnR,ECSnBlast);
-- bit 3 FEM enable
--ecs_parallel_write(SL+SL_CTRL_REG_ADDR,X"0000F810",ECSCLK,ECSAD,ECSnReady,ECsnCS1,ECSnADS,ECSWnR,ECSnBlast);
--ecs_parallel_write(SL+SL_CTRL_REG1_ADDR,X"00200F00",ECSCLK,ECSAD,ECSnReady,ECsnCS1,ECSnADS,ECSWnR,ECSnBlast);
--****** Re-configure PLL     ********
--ecs_parallel_write(SL+SL_TP_REG_ADDR,X"00440402",ECSCLK,ECSAD,ECSnReady,ECsnCS1,ECSnADS,ECSWnR,ECSnBlast);
--wait for 5 us;
--ecs_parallel_write(SL+SL_TP_REG_ADDR,X"00441404",ECSCLK,ECSAD,ECSnReady,ECsnCS1,ECSnADS,ECSWnR,ECSnBlast);
--wait for 5 us;
--ecs_parallel_write(SL+SL_TP_REG_ADDR,X"00020000",ECSCLK,ECSAD,ECSnReady,ECsnCS1,ECSnADS,ECSWnR,ECSnBlast);
--wait for 20 us;
--ecs_parallel_write(PP0+VELO_PHI_REORDER_STRIP_CNT_REG_ADDR,X"00000111",ECSCLK,ECSAD,ECSnReady,ECsnCS1,ECSnADS,ECSWnR,ECSnBlast);
--ecs_parallel_read(PP0+VELO_PHI_REORDER_STRIP_CNT_REG_ADDR,data_received,ECSCLK,ECSAD,ECSnReady,ECsnCS1,ECSnADS,ECSWnR,ECSnBlast);
--
--ecs_parallel_write(PP0+VELO_CLUSTER_PARA_REG_ADDR+0,X"00004034",ECSCLK,ECSAD,ECSnReady,ECsnCS1,ECSnADS,ECSWnR,ECSnBlast);
--ecs_parallel_write(PP0+VELO_CLUSTER_PARA_REG_ADDR+4,X"00404034",ECSCLK,ECSAD,ECSnReady,ECsnCS1,ECSnADS,ECSWnR,ECSnBlast);
--ecs_parallel_write(PP0+VELO_CLUSTER_PARA_REG_ADDR+8,X"00804034",ECSCLK,ECSAD,ECSnReady,ECsnCS1,ECSnADS,ECSWnR,ECSnBlast);
--ecs_parallel_write(PP0+VELO_CLUSTER_PARA_REG_ADDR+12,X"00C04034",ECSCLK,ECSAD,ECSnReady,ECsnCS1,ECSnADS,ECSWnR,ECSnBlast);
--ecs_parallel_write(PP0+V
--ecs_parallel_write(PP0+V
--ecs_parallel_write(PP0+V
--ecs_parallel_write(PP0+V
--ecs_parallel_write(PP0+V
--****** PP_CTRL_REGISTEF
--0..1 scaling mode
--2 pedestal enable
--3 pedestal update
--4 reorder enable
--5 FIR enable
--6 Data Gen enable
--7 Ber
--8 CM enabel
--9 ZS enable
--10 Phi reorder
--11 R reorder

--
--ecs_parallel_write(PP0+P
--ecs_parallel_write(PP1+P
--ecs_parallel_write(PP2+P
--ecs_parallel_write(PP3+P
--****** reset adc_clk_gen r
--ecs_parallel_write(PP0+A
```

Figure 6: The ECS Master is a flow chart that allows to issue sequential ECS access to memories and registers.

1. At time 0 the registers and memories are initialized with the default values given by the .hex file contents for RAM and ROM. It is also important to get the memory contents to valid values such that running the first 1 μs is not generating many warnings.

2. After the PLLs have locked (300 ns), the reset is removed and a one clock cycle long reset is issued again.

3. After 1 μs the memory initialization sets all ECS accessible RAM and ROM to the initial state corresponding to the configure file settings. Also the input stimulus (eg FE-models) are set to stimulate the test bench.

4. Ready to run !

9.2 ModelSim initialization

At the startup of ModelSim a script called **start.do** is called which allows to make some customization settings. It is also used to run the test bench for 2μs before the memory initialization is applied.

10 Command Line Compilation (CLI) scripts

There is a strong need of automatization of the design compilation. The time it takes to synthesize and place and route one FPGA can easily exceed 30 minutes. Precision and Quartus provide a tcl interface on a command line. These are very helpful to automate the design flow. The scrip `<hdl_root>\CLI_compile\CLI.bat` is a windows executable batch file which permits to make project assignments by calling the Quartus tcl scripts (see 4), to synthesize with Precision, to Place and Route (also synthesis if required) for Quartus and to convert the programming files to get the TELL1 pof file. The **CLI.bat** is writing a logfile.txt to report on the tasks processed.

11 Built number, date and time script

To obtain automatic version, date and time information accessible via ECS for each compilation, the CLI.bat chapter 10 calls a version script called `version_precision.tcl` or `version_quartus.tcl` to set the date, time and built number constants. The package that is creates can be found in the library `common_TELL1_lib` and the package name is `date_and_time`. The version scripts are located in the CLI directory but also a copy in the quartus_tcl directory.

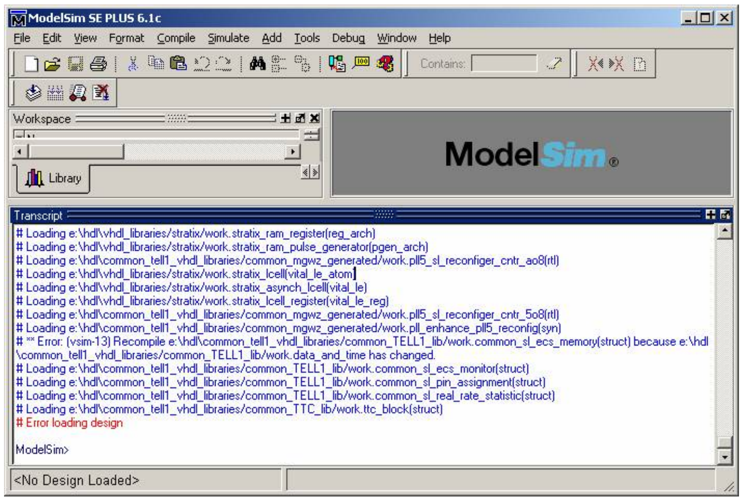
12 Solving problems

12.1 Inconsistent design after compilation

If Modelsim reports to have inconsistant libraries and requires recompilation (loading error. Complete recompilation of the design can be forced by setting $Tasks \hookrightarrow SetGenerateAlways$ $Tasks \hookrightarrow SetCompileAlways$ (see figure 7,8). This is typically needed after editing a package in the design. In the newer version (7.2 FPGA Advantage this is much less needed)

12.2 ModelSim libraries recompilation

Changing ModelSim version always requires to recompile the standard libraries. There are only four libraries used specific to the Altera Stratix devices. The source files are given in the source directory called src for each libraries. These libraries are: altera_mf, stratix, sgate, and lpm. The libraries are located in: `<hdl_root>\vhdl_libraries`. The libraries can be recompiled from the ModelSim GUI (see Fig. 9 use $Compile \hookrightarrow Compile$. Indicate the destination library (stratix) at the Library selection, indicate the VHDL source files
`<hdl_root>\vhdl_libraries\stratix\src`
and select all source files. The options for the compilation can be checked, use the VHDL-


ModelSim SE PLUS 6.1c
File Edit View Format Compile Simulate Add Tools Debug Window Help
Contains:
Workspace
Library
ModelSim
Transcript

```
# Loading e:\hdl\vhdl_libraries/stratix/work.stratix_ram_register(reg_arch)
# Loading e:\hdl\vhdl_libraries/stratix/work.stratix_ram_pulse_generator(pgen_arch)
# Loading e:\hdl\common_tell1_vhdl_libraries/common_mgwz_generated/work.pll5_sl_reconfiger_cntr_ao8(rtl)
# Loading e:\hdl\vhdl_libraries/stratix/work.stratix_lcell(vital_le_atom)
# Loading e:\hdl\vhdl_libraries/stratix/work.stratix_asynch_lcell(vital_le)
# Loading e:\hdl\vhdl_libraries/stratix/work.stratix_lcell_register(vital_le_reg)
# Loading e:\hdl\common_tell1_vhdl_libraries/common_mgwz_generated/work.pll5_sl_reconfiger_cntr_5o8(rtl)
# Loading e:\hdl\common_tell1_vhdl_libraries/common_mgwz_generated/work.pll_enhance_pll5_reconfig(syn)
# ** Error: (vsim-13) Recompile e:\hdl\common_tell1_vhdl_libraries/common_TELL1_lib/work.common_sl_ecs_memory(struct) because e:\hdl
\common_tell1_vhdl_libraries/common_TELL1_lib/work.data_and_time has changed.
# Loading e:\hdl\common_tell1_vhdl_libraries/common_TELL1_lib/work.common_sl_ecs_monitor(struct)
# Loading e:\hdl\common_tell1_vhdl_libraries/common_TELL1_lib/work.common_sl_pin_assignment(struct)
# Loading e:\hdl\common_tell1_vhdl_libraries/common_TELL1_lib/work.common_sl_real_rate_statistic(struct)
# Loading e:\hdl\common_tell1_vhdl_libraries/common_TTC_lib/work.ttc_block(struct)
# Error loading design

ModelSim>
```

<No Design Loaded>

Figure 7: Modelsim creates an error at the loading process.

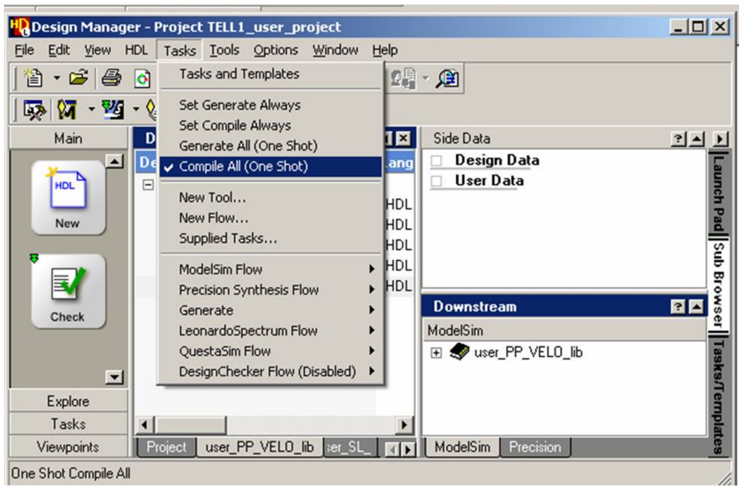
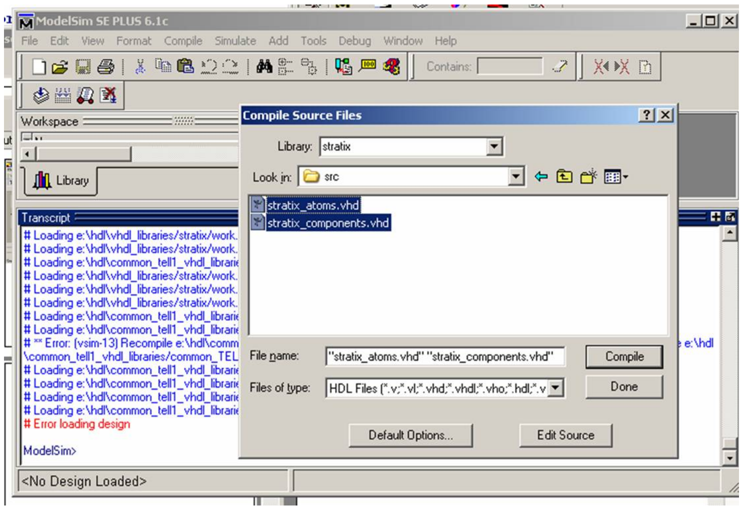


Figure 8: Select in HDL Designer to recompile the complete design.



```
# Loading e:\hdl\vhdl_libraries/stratix/work.
# Loading e:\hdl\vhdl_libraries/stratix/work.
# Loading e:\hdl\common_tell1_vhdl_librarie
# Loading e:\hdl\vhdl_libraries/stratix/work.
# Loading e:\hdl\vhdl_libraries/stratix/work.
# Loading e:\hdl\vhdl_libraries/stratix/work.
# Loading e:\hdl\common_tell1_vhdl_librarie
# Loading e:\hdl\common_tell1_vhdl_librarie
# ** Error: (vsim-13) Recompile e:\hdl\comm
\common_tell1_vhdl_libraries/common_TEL
# Loading e:\hdl\common_tell1_vhdl_librarie
# Loading e:\hdl\common_tell1_vhdl_librarie
# Loading e:\hdl\common_tell1_vhdl_librarie
# Loading e:\hdl\common_tell1_vhdl_librarie
# Error loading design

ModelSim>
```

Library: stratix

Look in: src

stratix_atoms.vhd
stratix_components.vhd

File name: "stratix_atoms.vhd" "stratix_components.vhd"

Files of type: HDL Files (*.v;*.vl;*.vhd;*.vhdl;*.vho;*.hdl;*.v

<No Design Loaded>

Figure 9: Compile the stratix library from the ModelSim GUI. Select the library and the source files from the src directory.

93 dialect.

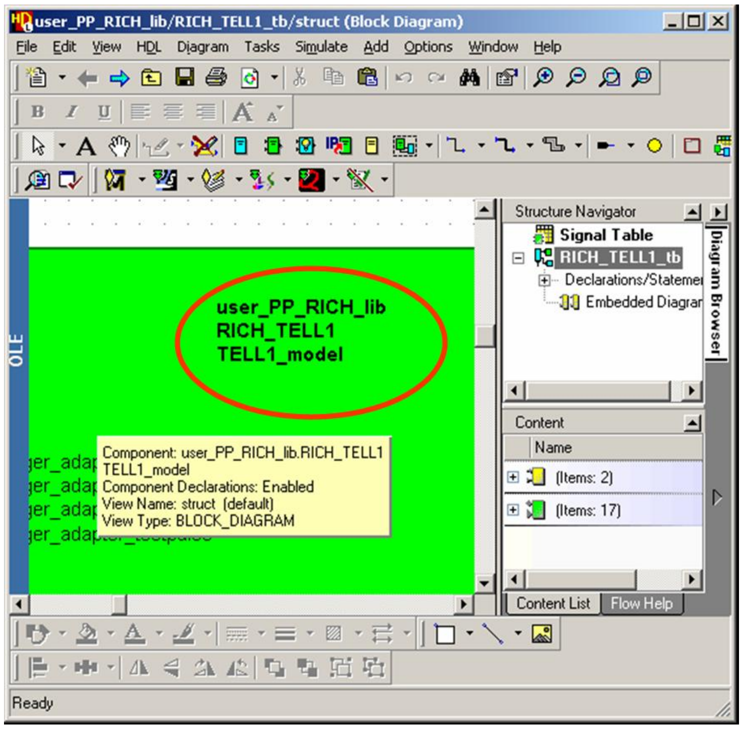
13 Create a new design (Quick guide)

Create new library mapping In `<hdl_root>\hdl_ini_files\TELL1_user_project.hdp` you can add your new library (called new_lib), this can also be done from the graphical interface from HDL Designer. Check that it uses the environment variables !

Copy the design from an example detector Open in HDL Designer your new project library (new_lib) and one that you can use as base (eg for optical receivers there is ST or RICH), select all components in user_PP_ST_lib and paste it into your new library (to paste something into a library, you need to select the library.)

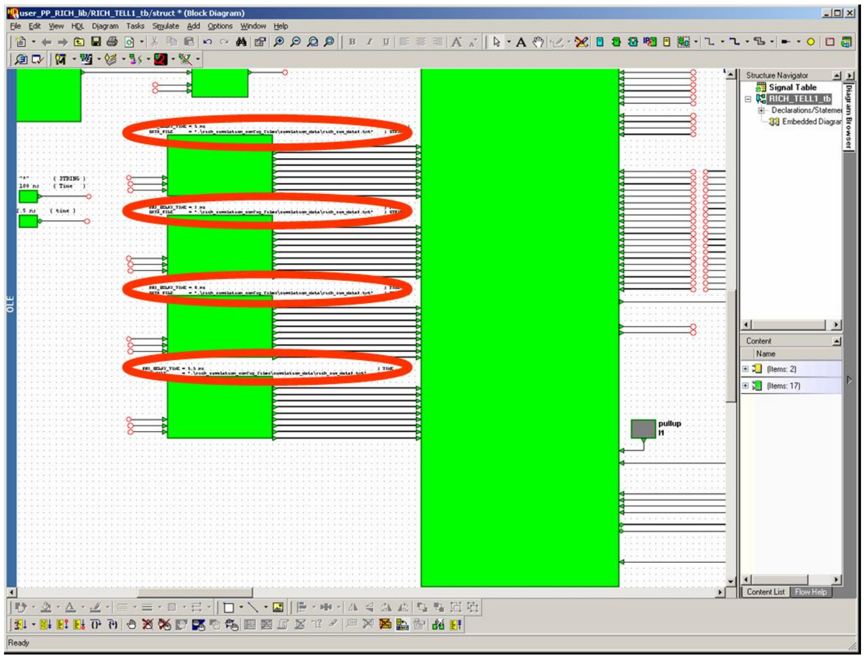
Rename Rename all copied components to create names corresponding to your new library name. HDL Designer asks you if it needs to replace all design units with this name, select no, otherwise the old (eg ST) library will also be affected. Warnings about c̈onverting old to new symbolsc̈an be ignored.
Now there is a test bench and all necessary components to build it, just make sure that in the new_lib all dependencies are to the new_lib components 10. You have to go into the hierarchy of the test bench and follow the PP_FPGA, change all names to the new_lib name.

Test bench stimulus Make sure you change the file name for the stimulus in the test bench, top level test bench,



Component: user_PP_RICH_lib.RICH_TELL1
TELL1_model
Component Declarations: Enabled
View Name: struct (default)
View Type: BLOCK_DIAGRAM

Figure 10: Make sure that all settings of the components point to the new library, eg here the new library is for RICH.



```
FE0_DELAY_TIME = 5 ns
DATA_FILE      = ".\rich_simulation_config_files\simulation_data\rich_sim_data4.txt"
FE1_DELAY_TIME = 3 ns
DATA_FILE      = ".\rich_simulation_config_files\simulation_data\rich_sim_data4.txt"
FE2_DELAY_TIME = 6 ns
DATA_FILE      = ".\rich_simulation_config_files\simulation_data\rich_sim_data4.txt"
FE3_DELAY_TIME = 5.5 ns
DATA_FILE      = ".\rich_simulation_config_files\simulation_data\rich_sim_data4.txt"
```

Figure 11: Change the stimulus file name for the four FE generators.

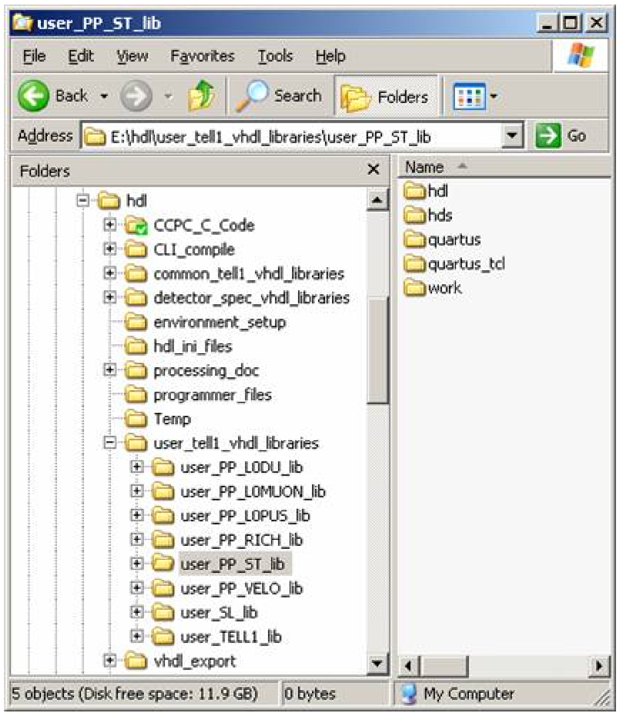


Figure 12: The figure shows the structure of the user directory - please keep this for your design in order to get the relative path settings correct.

New Quartus project To setup a new Quartus project copy quartus_tcl from eg the user_PP_ST_lib into your user library directory. You should also make a new Quartus directory and include the gen.tcl and the mem_ini folder. Configure the top.tcl in quartus_tcl with the required names for the project. Respect the same structure of the user library directory as shown in Fig. 12.

Convert programming file Make a new .cof file in `<hdl_root>\programmer_files` that is adapted to the name of the new project and uses the correct .sof files for the conversion.